Unicode编码

1. Unicode 编码来历

Unicode是为了解决传统的[字符编码](https://zh.wikipedia.org/wiki/%E5%AD%97%E5%85%83%E7%B7%A8%E7%A2%BC)方案的局限而产生的，例如[ISO 8859-1](https://zh.wikipedia.org/wiki/ISO/IEC_8859)所定义的字符虽然在不同的国家中广泛地使用，可是在不同国家间却经常出现不兼容的情况。很多传统的编码方式都有一个共同的问题，即容许电脑处理双语环境（通常使用[拉丁字母](https://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D)以及其本地语言），但却无法同时支持多语言环境（指可同时处理多种语言混合的情况）。

2.Unicode 编码方式

目前实际应用的统一码版本对应于[UCS-2](https://zh.wikipedia.org/wiki/UCS-2)，使用16[位](https://zh.wikipedia.org/wiki/%E4%BD%8D%E5%85%83)的编码空间。也就是每个字符占用2个[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)。这样理论上一共最多可以表示216（即65536）个字符。基本满足各种语言的使用。实际上当前版本的统一码并未完全使用这16位编码，而是保留了大量空间以作为特殊使用或将来扩展。上述16位统一码字符构成[基本多文种平面](https://zh.wikipedia.org/wiki/%E5%9F%BA%E6%9C%AC%E5%A4%9A%E6%96%87%E7%A8%AE%E5%B9%B3%E9%9D%A2)。最新（但未实际广泛使用）的统一码版本定义了16个[辅助平面](https://zh.wikipedia.org/wiki/%E8%BE%85%E5%8A%A9%E5%B9%B3%E9%9D%A2)，两者合起来至少需要占据21位的编码空间，比3字节略少。但事实上辅助平面字符仍然占用4字节编码空间，与[UCS-4](https://zh.wikipedia.org/wiki/UCS-4)保持一致。未来版本会扩充到ISO 10646-1实现级别3，即涵盖UCS-4的所有字符。UCS-4是一个更大的尚未填充完全的31位字符集，加上恒为0的首位，共需占据32位，即4字节。理论上最多能表示231个字符，完全可以涵盖一切语言所用的符号。

3.Unicode 兼容哪几种语言

大部分的语言都能兼容

Big 5 编码（大五码或五大码）

1.Big 5编码来历

Big5码因为当时[个人电脑](https://zh.wikipedia.org/wiki/%E5%80%8B%E4%BA%BA%E9%9B%BB%E8%85%A6)没有共通的内码，导致厂商推出的中文应用软件无法推广，并且与[IBM 5550](https://zh.wikipedia.org/wiki/IBM_5550)、[王安码](https://zh.wikipedia.org/w/index.php?title=%E7%8E%8B%E5%AE%89%E7%A2%BC&action=edit&redlink=1)等内码，彼此不能兼容；另一方面，台湾当时尚未推出中文编码标准。在这样的时空背景下，为了使台湾早日进入信息时代，所采行的一个项目；同时，这个项目对于以台湾为核心的亚洲繁体汉字圈也产生了久远的影响

2.Big 5 编码方式

Big5码是一套双字节字符集，使用了双八码存储方法，以两个字节来安放一个字。第一个字节称为“高位字节”，第二个字节称为“低位字节”。“高位字节”使用0X81-0XFE，“低位字节”使用了0XA1-0XFE。

3.Big 5 兼容哪几种语言

兼容中国繁体字

GB 2312(GB 2312-80)

1.GB 2312编码来历

GB 2312的出现，基本满足了汉字的计算机处理需要，它所收录的汉字已经覆盖中国大陆99.75%的使用频率。但对于[人名](https://zh.wikipedia.org/wiki/%E4%BA%BA%E5%90%8D)、[古汉语](https://zh.wikipedia.org/wiki/%E5%8F%A4%E6%B1%89%E8%AF%AD)等方面出现的[罕用字](https://zh.wikipedia.org/wiki/%E7%BD%95%E7%94%A8%E5%AD%97)和[繁体字](https://zh.wikipedia.org/wiki/%E7%B9%81%E9%AB%94%E5%AD%97)，GB 2312不能处理，因此后来[GBK](https://zh.wikipedia.org/wiki/GBK)及[GB 18030](https://zh.wikipedia.org/wiki/GB_18030)汉字字符集相继出现以解决这些问题。

2.GB 2312 编码方式

在使用GB 2312的程序通常采用[EUC](https://zh.wikipedia.org/wiki/EUC)储存方法，以便兼容于[ASCII](https://zh.wikipedia.org/wiki/ASCII)。这种格式称为**EUC-CN**。

每个汉字及符号以两个[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)来表示。第一个字节称为“高位字节”，第二个字节称为“低位字节”。“高位字节”使用了0xA1–0xF7（把01–87区的区号加上0xA0），“低位字节”使用了0xA1–0xFE（把01–94加上0xA0）。 由于一级汉字从16区起始，汉字区的“高位字节”的范围是0xB0–0xF7，“低位字节”的范围是0xA1–0xFE，占用的码位是72\*94=6768。其中有5个空位是D7FA–D7FE。

3.GB 2312兼容哪几种语言

标准简体中文

4.GB 2312编码的问题

基本满足了汉字的计算机处理需要，它所收录的汉字已经覆盖中国大陆99.75%的使用频率。但对于人名、古汉语等方面出现的罕用字和繁体字，GB 2312不能处理，因此后来GBK及GB 18030汉字字符集相继出现以解决这些问题

ASCII 码

1. ASCII码的来历

在计算机中，所有的数据在存储和运算时都要使用二进制数表示（因为计算机比较傻，只有0和1两位数的二进制比较适合于它使用），同样的，像a、b、c、d这样的52个字母（包括大写）、以及0、1、2等数字还有一些常用的符号（例如\*、#、@等）在计算机中存储时也要使用二进制数来表示，而具体用哪个数字表示哪个符号，当然每个人都可以约定自己的一套（这就叫编码），而大家如果要想互相通讯而不造成混乱，那么大家就必须使用相同的编码规则，于是美国有关的标准化组织就出台了所谓的[ASCII](https://www.baidu.com/s?wd=ASCII&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nAm4nH0dn1ubPWwbnHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDknjnYPHTLnW6YrHmdPjnLrf)编码，统一规定了上述常用符号用哪个二进制数来表示。

1. ASCII码的编码方式

ASCII码一共规定了128个字符的编码，比如空格“SPACE”是32（二进制00100000），大写的字母A是65（二进制01000001）。这128个符号（包括32个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的1位统一规定为0。

1. ASCII码兼容哪几种语言

适用于现代英文和其他西欧语言

1. ASCII码的问题

由于编码范围有限,所以无法支持多种地区的语言,后来就产生了各种其它的编码方案

UTF-8 码

1. UTF-8码的来历

为创建良好的[字节串编码系统](https://zh.wikipedia.org/w/index.php?title=%E4%BD%8D%E5%85%83%E7%B5%84%E4%B8%B2%E7%B7%A8%E7%A2%BC%E7%B3%BB%E7%B5%B1&action=edit&redlink=1" \o "字节串编码系统（页面不存在）)以供多[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82" \o "字节)字符集使用，开始了一个正式的研究。[ISO/IEC 10646](https://zh.wikipedia.org/wiki/%E9%80%9A%E7%94%A8%E5%AD%97%E7%AC%A6%E9%9B%86)的初稿中有一个非必须的附录，名为UTF。当中包含了一个供32[比特](https://zh.wikipedia.org/wiki/%E4%BD%8D%E5%85%83)的[字符](https://zh.wikipedia.org/wiki/UCS)使用的字节串编码系统。这个编码方式的性能并不令人满意，但它提出了将0-127的范围保留给ASCII以兼容旧系统的概念。[X/Open](https://zh.wikipedia.org/wiki/X/Open)委员会X0JIG开始寻求一个较佳的编码系统。[Unix系统实验室](https://zh.wikipedia.org/wiki/Unix%E7%B3%BB%E7%BB%9F%E5%AE%9E%E9%AA%8C%E5%AE%A4" \o "Unix系统实验室)（USL）的Dave Prosser为此提出了一个编码系统的建议。它具备可更快速实现的特性，并引入一项新的改进。其中，7[比特](https://zh.wikipedia.org/wiki/%E4%BD%8D%E5%85%83)的[ASCII](https://zh.wikipedia.org/wiki/ASCII)符号只代表原来的意思，所有多字节序列则会包含第8[比特](https://zh.wikipedia.org/wiki/%E4%BD%8D%E5%85%83)的符号，也就是所谓的[最高有效比特](https://zh.wikipedia.org/wiki/%E6%9C%80%E9%AB%98%E6%9C%89%E6%95%88%E4%BD%8D%E5%85%83)。

1. UTF-8码的编码方式

UTF-8是UNICODE的一种变长度的编码表达方式《一般UNICODE为双位元组（指UCS2）》，它由[肯·汤普逊](https://zh.wikipedia.org/wiki/%E8%82%AF%C2%B7%E6%B1%A4%E6%99%AE%E9%80%8A" \o "肯·汤普逊)（Ken Thompson）于1992年创建，现在已经标准化为RFC 3629。UTF-8就是以8位为单元对UCS进行编码，而UTF-8不使用大尾序和小尾序的形式，每个使用UTF-8存储的字符，除了第一个字节外，其余字节的头两个比特都是以"10"开始，使文字处理器能够较快地找出每个字符的开始位置。

1. UTF-8编码兼容语言

与Unicode编码兼容一样的语言

1. UTF-8编码的问题

UTF-8在中文字符处理时，一般都是3个字节，这样相比GBK，UTF-16等编码，会更浪费空间(但是针对英文又有天然的优势-单字节。而且因为每个字符使用不同数量的字节编码，所以寻找串中第N个字符是一个O(N)复杂度的操作 — 即，串越长，则需要更多的时间来定位特定的字符。同时，还需要位变换来把字符编码成字节，把字节解码成字符。

UTF-16编码

1. UTF-16 码的来历

Unicode编码中，最常用的字符其实是0-65535，因此针对这点产生了UTF-16方案。

1. UTF-16编码方式

UTF-16将0–65535范围内的字符编码成2个字节，超过这个的用4个字节编码。(因此基本可以认为是双字节的)。UTF-16是完全对应于UCS-2的，即把UCS-2规定的代码点通过Big Endian或Little Endian方式直接保存下来。所以UTF-16采用2个字节来存储Unicode。UTF-16也可以表示UCS-4的部分字符，所以UTF-16也采用4个字节来存储Unicode。

1. UTF-16兼容语言

与Unicode编码兼容一样的语言。

几种编码的转换

一般来说UTF-8可应用于大多数场景，尤其是互联网上，而中文编码主要使用GBK编码，因此这就有了GBK、GB2312和UTF-8的相互转换需求。但要注意， GBK、GB2312等与UTF8之间都必须通过Unicode编码才能相互转换 ，也即： GBK、GB2312 ---> Unicode ---> UTF-8 UTF8 ---> Unicode ---> GBK、GB2312

比如拿汉字的“字”来举例，

“字“的表示

二进制表示

16进制

10进制

字符串表示

GBK

11010111 11010110

D7 D6

55254

字

Unicode

01011011 01010111

5B 57

23383

Utf-8

11100101 10101101 10010111

E5 AD 97

15052183

瀛

同样，对于词语“中国.北京”，转换成UTF-8表示为：涓浗.鍖椾含。

你要问我怎么转换的？原理什么的就不介绍了，在Windo平台下，通过一些Win API可以将GBK和Utf-8经过Unicode层进行相互转换。Linux下也是这样。另外由于各个编码规则都是确定的，因此各字符也就有了一种对应关系，这就是查表法也成为实现编码转换的一种方式。

GBK和UTF-8编码的检测

上面简单介绍了各种编码以及他们的相互转换，但如果一串字符串，能够检测出其编码是什么吗？ 比如对于“中国.北京”，检测出其为GBK或GB2312，而对于

”涓浗.鍖椾含“检测出其为UTF-8。那么对于一种确定的检测算法，是否保证肯定能检测到该字符串的真正编码方式吗？或者对于一个网页内容来说，很可能既存在GBK编码内容，也有UTF-8内容，那网页应该怎么显示（好像是检测编码，哪种编码出现的频率多，就使用哪种编码显示，臆测，不懂）？

再举一个例子： “谢谢”，其UTF-8表示是“璋㈣阿”。从我的实现中，不论对于GBK编码的“谢谢”，还是UTF-8的“璋㈣阿”，都 检测出是UTF-8编码 。而我也不是第一个遇到这种情况的人，比如 孟岩

他老人家也遇到过检测失败的问题，不过他是在Ruby语言下发现的问题，而我是在C/C++实现中出现的检测失败。

好吧，说到这里我也不知道说的是什么了，总之就是，

各编码规则从应用角度来看，还挺复杂；

编码转换和编码检测有时也不靠谱；

附上我的测试代码，请高手指点我的实现是否有问题，谢谢！

void GetText(bool isUtf8, std::vector<<span>std::string>& vecStr) 2 { 3 const char\* filename = isUtf8 ? "utf8.txt" : "gbk.txt"; 4 std::ifstream ifs(filename); 5 assert(!ifs.fail()); 6 7 while(!ifs.eof()) 8 { 9 std::string str;10 ifs >> str;11 vecStr.push\_back(str);12 }13 14 ifs.close();15 }16 17 int main()18 {19 std::vector<<span>std::string> uVecStr;20 std::vector<<span>std::string> gVecStr;21 GetText(true,uVecStr);22 GetText(false,gVecStr);23 24 std::cout<<"--------------utf8--->gbk----------------\n";25 for (std::vector<<span>std::string>::iterator itr = uVecStr.begin();itr!=uVecStr.end();++itr)26 {27 std::cout<<\*itr<<"\t"<<ConvertUtf8ToGBK(\*itr)<<"\t"<<IsTextUTF8((\*itr).c\_str(),(\*itr).size())<<"\t"<<IsTextUTF8(\*itr)<<"\n";28 }29 30 std::cout<<"--------------gbk--->utf8----------------\n";31 for (std::vector<<span>std::string>::iterator itr = gVecStr.begin();itr!=gVecStr.end();++itr)32 {33 std::cout<<\*itr<<"\t"<<ConvertGBKToUtf8(\*itr)<<"\t"<<IsTextUTF8((\*itr).c\_str(),(\*itr).size())<<"\t"<<IsTextUTF8(\*itr)<<"\n";34 }35 return 0;36 }

上面测试代码是分别从两个文本文件中读取相同内容，一个文件内容是UTF-8字符串，然后转换为GBK编码，并检测源字符串编码方式；而另一个文件内容是GBK编码，然后转换为UTF-8编码，并检测源字符串编码方式。文件内容如下：

谢谢

你好

中国.北京

输出内容如下：

你好 浣犲ソ 0

0谢谢 璋㈣阿 1 1

​--------------utf8--->gbk----------------

锘胯阿璋 ?谢谢 1 1

​浣犲ソ 你好 1 1

​涓浗.鍖椾含 中国.北京 1 1

​--------------gbk--->utf8----------------

谢谢 璋㈣阿 1 1

​你好 浣犲ソ 0 0

​中国.北京 涓浗.鍖椾含 0 0​​